

Hybrid Discrete-Continuous Markov Decision Processes

Zhengzhu Feng

Department of Computer Science
University of Massachusetts
Amherst, MA 01003-4610
fengzz@cs.umass.edu

Abstract

This paper proposes a Markov decision process (MDP) model that features both discrete and continuous state variables. We extend previous work by Boyan and Littman on the mono-dimensional time-dependent MDP to multiple dimensions. We present the principle of lazy discretization, and piecewise constant and linear approximations of the model. Having to deal with several continuous dimensions raises several new problems that require new solutions. In the (piecewise) linear case, we use techniques from partially-observable MDPs (POMDPs) to represent value functions as sets of linear functions attached to different partitions of the state space.

Introduction

Because of communication limitations, remote spacecraft and rovers need the ability to operate autonomously. For instance, the Mars Exploration Rovers (MER) will communicate with the ground only twice per Martian day and must operate autonomously the rest of the time. Moreover, the surfaces of planets are very uncertain environments. In the case of Mars, there is uncertainty about the terrain, the meteorological conditions, and the state of the rover itself (position, battery charge, solar panels, component wear, etc.), resulting in a great deal of uncertainty in the duration, energy consumption and outcome of the rover's actions (Bresina *et al.* 2002). The need for autonomy and robustness in the face of uncertainty will grow as rovers become more capable and as missions explore more distant planets.

Planning systems that have been developed for planetary rovers and other NASA applications typically use a deterministic model of the environment and action effects (Muscettola *et al.* 1998; Jónsson *et al.* 2000; Estlin *et al.* 2002). Given a pre-specified set of goals, they produce a deterministic sequence of actions that attempts to achieve the goals, assuming nominal conditions. They do not model the uncertainty in the domain, but instead rely on replanning to handle unexpected events. This straightforward approach presents several drawbacks: (i) it does not address the compromise between the value of a goal and the risk attached to it; (ii) it does not choose good branch points: waiting

Richard Dearden* **Nicolas Meuleau†**

Rich Washington‡

NASA Ames Research Center, Mail Stop 269-3
Moffet Field, CA 94035-1000
{dearden, nmeuleau, richw}@email.arc.nasa.gov

for a failure of the nominal plan is a poor strategy, since it could be too late to do anything interesting; (iii) it does not identify the benefits of "set-up actions," that is, actions that must be integrated into the nominal plan only because they could be useful if we had to revise our plans in the course of execution (for instance, putting a spare tire in the trunk before going on a car trip).

Decision theory is a principled framework for reasoning about uncertainty, rewards, and costs (Blythe 1999; Boutilier, Dean, & Hanks 1999). It avoids the three pitfalls of previous approaches: (i) it makes optimal trade-offs between the value of goals and plans, and the risks associated with them; (ii) it selects optimal branch points;¹ (iii) it captures the necessity of performing set-up actions each time there is benefit in doing so. Therefore, decision theory seems particularly suited to respond to the need for autonomy in NASA applications. However, there are many obstacles to a direct application of decision-theoretic algorithms such as dynamic programming (DP) (Howard 1970; Puterman 1994) to real domains such as planetary exploration rovers (Bresina *et al.* 2002). In this paper, we focus on one of these difficulties: the existence of continuous state variables.

A characteristic of many of NASA application domains is the existence of continuous state variables such as time, battery levels, location, and available memory. Most of them represent resources that constrain the planning problem. Moreover, most of the uncertainty in the domain results from the effect of actions on these variables. In the Mars rover domain, the biggest sources of uncertainty are the duration and energy consumption of actions and the storage space that pictures will require after compression. In contrast, the control framework is not completely continuous because decisions are made at discrete decision steps. Formally, the problem is that of a discrete-step decision model, such as a Markov Decision Process (MDP) (Puterman 1994), with several continuous state variables. The continuous variables make the state space continuously infinite.

¹It is commonly believed that decision theory is limited to searching for an optimal policy, that is, a complete conditional plan with one "branch" for each possible situation that could be encountered at execution. However, recent work has shown that it is possible to use a decision theoretic approach to find optimal plans of other types, such as conformant plans (Hyafil & Bacchus 2003) and k -contingency plans (Meuleau & Smith 2003).

*Research Institute for Advanced Computer Science.

†QSS Group Inc.

‡RIACS.

nite and prevent a direct use of classical solution techniques such as DP.

The common practice to deal with continuous state variables in MDPs is either to use function approximators such as artificial neural networks (Bertsekas & Tsitsiklis 1996; Sutton & Barto 1998), or to discretize the continuous state space more or less naively, which does not scale well to multiple dimensions. Munos and Moore (Munos 2000; Munos & Moore 2002) propose a formal model of a continuous MDP, and the theoretical foundations and algorithms for discretizing it. However, their model is a deterministic MDP where actions must be applied over continuous durations. The non-determinism in action outcomes results only from the discretization. This does not fit the problem of planetary rover planning that uses discrete global commands such as “drive from lander to rock 1”, and is intrinsically rife with exogenous uncertainty (Bresina *et al.* 2002). In another approach, Boyan and Littman have proposed a model of time dependent MDP (TMDP) that features one uncertain continuous state variable representing time (Boyan & Littman 2000). They approximate this model using piecewise linear approximations and develop algorithms that are able to efficiently solve a collection of benchmark problems.

In this paper, we extend Boyan and Littman’s TMDP by introducing *hybrid* MDPs (HyMDPs) that feature several continuous state variables. We then introduce the principle of lazy discretization, which is implicit in Boyan and Littman’s work. This idea is implemented in a *functional* dynamic programing algorithm that backs up piecewise constant approximations of continuous (value) functions, from discrete state to discrete state. This algorithm is shown to compare favorably to the naive discretization often used in the literature. Further, we extend this algorithm to piecewise linear rewards and value functions. There are several problems that appear when we want to generalize Boyan and Littman’s work from one dimension to multiple dimensions. The problems arise in representing and manipulating partitions of the multidimensional continuous state space. We propose original solutions to these problems using concepts from computational geometry and techniques from the theory of partially observable MDPs (POMDPs).

The Continuous Model

Model Definition

We propose a model of a hybrid MDP (HyMDP) that features both continuous and discrete state variables.

Discrete state variables: Let $x \in X$ represent the discrete part of a state. X is supposed finite. x may be a vector of discrete features with some structure. We do not need that kind of representation in this work but do not exclude it. If there is some structure in the discrete features, then our algorithms can be modified to exploit it in the same way as structured DP (Boutilier, Dearden, & Goldszmidt 2000). We call x the *discrete state* of the process.

Continuous state variables: For all $i \in \{1, 2, \dots, l\}$, θ_i represents a continuous state variable taking its value in the

compact interval Θ_i of \mathbb{R} (or a finite union of such intervals). We call $\theta = (\theta_1, \theta_2, \dots, \theta_l) \in \Theta = \bigotimes_{i=1}^l \Theta_i$ the *continuous state* of the process. This work is motivated by problems where the continuous variables represent almost exclusively resources such as the time and energy available to a rover. In common rover models, the rover position is represented as a discrete *location*, and movement actions are assumed either to lead to the destination (taking random time and energy), or to fail and leave the rover in an unknown location where back-up plans are used. However, there is theoretically no obstacle to applying our approach to any kind of continuous variable.

States: A (Markov) state s is an element of $S = X \times \Theta$. Note that we can easily extend the model so that the domain of each continuous variable varies from one discrete state x to another.

Actions: A is the finite set of actions available in each state. Again, we can easily extend the model so that the set of available actions varies from one state to another.² This is necessary in Mars rovers domain because there may be minimum levels of resources required to start actions (Bresina *et al.* 2002).

State transitions: The transition probability of the MDP is represented by the conditional probability distribution $T(s' | s, a)$ on the arrival state $s' = (x', \theta')$, given the starting state $s = (x, \theta)$ and action a . This distribution must reflect the possible monotonicity of some resource. For instance, if there is only one continuous variable θ_1 representing the time remaining, which can never increase, then $T((x', \theta') | (x, \theta), a) = 0$ when $\theta' > \theta$. It is convenient to decompose the (joint) transition distribution T into:

- a discrete marginal probability distribution on the arrival discrete state: $T_x(x' | s, a) \in [0, 1]$ is the probability that executing action a in state s results in discrete state x' . For all (s, a) , $\sum_{x \in X} T(x | s, a) = 1$;
- a conditional continuous distribution $T_\theta(\theta' | s, a, x')$ on the arrival continuous state θ' , given the starting state, action and arrival discrete state. This one may in turn be decomposed in a sequence of continuous conditionals: $T_{\theta_i}(\theta'_i | s, a, x', \theta'_{<i})$ where $\theta'_{<i} = (\theta'_1, \dots, \theta'_{i-1}) \in \Theta_{<i} = \bigotimes_{j=1}^{i-1} \Theta_j$.

Rewards: A reward function $R(s, a, s')$ represents the reward for a transition from state s to state s' under action a . A priori, the reward functions may be any continuous function. In practice, we have to assume some particular shape to develop practical algorithms.

Bellman Equations

We are interested in maximizing the expected reward of a plan over a finite horizon of H decision-steps. The flat Bell-

²To apply the solution techniques proposed hereafter, we have to assume that, for each $x \in X$, the set of actions available in (x, θ) is piecewise constant (w.r.t. θ).

man equation is

$$V_n(s) = \max_{a \in A} \left[\int_{s' \in S} T(s' | s, a) (R(s, a, s') + V_{n+1}(s')) ds' \right],$$

where $V_n(s)$ is the optimal reward from state s at decision step $n \in \{0, 1, \dots, H-1\}$, and $V_H(s) = 0$ for all s . It can be decomposed in the following way:

$$V_n(s) = \max_{a \in A} [Q_n(s, a)] ; \quad (1)$$

$$Q_n(s, a) = \sum_{x' \in X} T_x(x' | s, a) U_n(s, a, x') ; \quad (2)$$

$$U_n(s, a, x') = \int_{\theta' \in \Theta} T_\theta(\theta' | s, a, x') (R(s, a, s') + V(s')) d\theta' ; \quad (3)$$

with the conventions $s = (x, \theta)$ and $s' = (x', \theta')$. Equation (3) can in turn be decomposed by introducing a sequence of intermediary value functions

$$U_n^i(s, a, x', \theta'_{<i}) = \int_{\theta'_i \in \Theta_i} T_{\theta_i}(\theta'_i | s, a, x', \theta'_{<i}) U_n^{i+1}(s, a, x', \theta'_{<i+1}) d\theta'_i \quad (4)$$

for all $i \in \{1, 2, \dots, l-1\}$, and

$$U_n^l(s, a, x', \theta'_{<l}) = \int_{\theta'_l \in \Theta_l} T_{\theta_l}(\theta'_l | s, a, x', \theta'_{<l}) (R(s, a, s') + V_{n+1}(s')) d\theta'_l . \quad (5)$$

The main obstacle to the application of existing MDP algorithms (Puterman 1994; Sutton & Barto 1998) to our hybrid model is the convolution in step (3) (or (4) and (5)), and the issue of storing in memory continuous multidimensional value functions.

Discrete Approximations

Naive versus Lazy Discretization

A naive approach is to discretize the continuous variables by imposing a grid (uniform or not) over the state space. From this discretized model of states, discretized models of action effects (transition probabilities and rewards) are built. The problem is then solved as any other discrete problem. This straightforward approach becomes extremely costly as the dimensionality increases. This is because the size of the discrete state space increases exponentially with the number of continuous variables.

Figure 4 represents the optimal value function from the initial state of a simple Mars rover problem as a function of two continuous variables: the time and energy remaining. The shape of this value function is characteristic of the Mars rover domain, as well as other domains featuring a finite set of goals with positive utility and resource constraints. Noticeably, it includes vast *plateau* regions where the expected reward is nearly constant. These represent

regions of the state space where the optimal policy is the same, and the probability distribution on future history induced by this optimal policy is nearly constant. Clearly, there is no benefit in using a fine discretization grid in these regions of the state-space, and in fact there is a large cost added for doing so. In other regions of the state space—for instance, the curved hump where there is more time and energy available—a fine discretization helps increase the quality of the value function. This observation motivates the approach described below.

Following Boyan and Littman, we propose to solve HyMDPs using *lazy discretization*, whose principle is the following: instead of imposing a uniform discretization of states and deducing a discretization of action effects from it, we do the inverse. We start by constructing a discrete model of action effects on continuous variables, possibly using the same grid size (in each dimension) as in the naive approach. In the Mars rover domain, it consists of discretizing the resource consumption of actions (which can easily handle dependencies between different resources). Then, assuming that immediate rewards are piecewise constant functions of the continuous variables, a minimal discretization of the state space is computed at the same time as DP is performed (that is, backward from the planning horizon to the initial time). The value function at each step is represented by a piecewise constant function of the continuous variables, and the set of pieces over which it is defined is kept minimal to retain only the significant differences between states, given the discrete model of action effects. States matching the same piece of value function

- have the same optimal plan/policy,
- generate the same probability distribution on future history, in terms of actions performed, rewards received, and pieces of value functions traversed under this optimal policy (assuming the discrete model of action effects).

Given a fixed discretization step in each dimension, lazy discretization attains exactly the same accuracy as naive discretization, but it avoids all redundant computation. If the problem structure is such that a fine grid is required everywhere, lazy discretization will discretize the state space uniformly. In this case, it will be outperformed by naive discretization, which is a more direct computation. However, we argue that many problems (notably in the rover domain) do not require a fine grid everywhere and thus favor the lazy approach.

Functional Dynamic Programming

Formally, we assume that:

- The discrete marginals on the arrival discrete state $T_x(x' | s, a)$ are *piecewise constant* (w.r.t. θ).
- The conditional probability distributions on continuous state variables $T_\theta(\theta' | s, a, x')$ are *discrete* with a *finite* set of possible outcomes. As in Boyan and Litmann's TMDPs, we distinguish two different ways to discretize these probability distributions. For each action a and each continuous state variable θ_i that it impacts, the effect of a on θ_i may be:

absolute: we assume that the set of possible final values of θ_i is finite and constant for all starting states. Equation (4) becomes:

$$U_n^i(s, a, x', \theta'_{<i}) = \sum_{\theta'_i \in \Theta_i^a} T_{\theta_i}(\theta'_i | s, a, x', \theta'_{<i}) U_n^{i+1}(s, a, x', \theta'_{<i+1}), \quad (6)$$

where Θ_i^a is the finite set of values that θ_i may take after executing action a . We also assume that the probability of each discrete outcome $T_{\theta_i}(\theta'_i | s, a, x')$ is *piecewise constant* (w.r.t. θ), and thus all conditionals that derive from it are also;

relative: the discretization does not concern the set of final values of θ_i , but the set of possible variations of θ_i resulting from action a . Denoting the variation of θ_i as $\delta_i = \theta'_i - \theta_i$, we assume that δ_i takes its value in the finite set Δ_i^a with probability 1. In other words, the set of possible values for θ'_i varies continuously (and “linearly”) with the starting state s . Equation (4) becomes:

$$U_n^i(s, a, x', \theta'_{<i}) = \sum_{\delta_i \in \Delta_i^a} T_{\delta_i}(\delta_i | s, a, x', \theta'_{<i}) U_n^{i+1}(s, a, x', \theta'_{<i+1}). \quad (7)$$

Here too we need to assume that the probability of each discrete value for δ_i , $T_{\delta_i}(\delta_i | s, a, x')$, is *piecewise constant* (w.r.t. θ), and thus all conditionals derived from it are also.

An action may have an absolute effect on some variables and a relative effect on others. In the planetary rover domain for instance, the action “wait until the battery is full” has a relative effect on time and an absolute effect on energy. Conversely, the action “wait for the next communication window” has an absolute effect on time and a relative effect on energy.

To specify the shape of the immediate reward function R , we use the following result:

Property 1 *Under the hypotheses above, if the reward function $R(s, a, s')$ is piecewise polynomial of order k (w.r.t. (θ, θ')), then the value function at each step $V_n(s)$ is also piecewise polynomial of order k (w.r.t. θ).*

Based on this result, we propose piecewise constant and piecewise linear approximations of HyMDPs ($k = 0$ and 1). We then solve these approximations using a form of *functional dynamic programing*. This algorithm performs a finite number of back-ups, but each back-up involves an infinite number of states. Instead of backing up scalars that represent the value of a single state as in the classical DP, it backs up piecewise constant or linear approximations of the value of an infinite number of (continuous) states.³ By keeping the number of pieces as small as possible, we implement the idea of lazy discretization.

This is a direct generalization of Boyan and Littman’s work. However, there is a major difficulty that arises when

³The same principle of functional DP is used in the classical solution techniques for POMDPs (Kaelbling, Littman, & Cassandra 1998; Cassandra, Littman, & Zhang 1997) and Boyan and Littman’s TMDPs (Boyan & Littman 2000).

we move from a uni-dimensional model as the TMDP to multiple dimensions. In a uni-dimensional framework, the piecewise constant or linear value functions use pieces that are simple intervals of the real line. These partitions are thus relatively easily to store and manipulate. In multiple dimensions, a minimal partition may contain pieces that are not hyper-rectangles, and thus it is costly and difficult to maintain. In the piecewise constant model, we remedy this problem by computing an almost minimal partition that uses only rectangular pieces. In the piecewise linear model, a rectangular decomposition is not possible, since the pieces of a minimal partition may have complex polyhedral shapes. Here, we borrow ideas from POMDP solution techniques to propose efficient algorithms. These approaches for piecewise constant and piecewise linear models are detailed in the next sections.

Piecewise Constant Model

Foundations

If actions have been discretized as explained above and (immediate) rewards are piecewise constant, then the value function at each step is piecewise constant. In this case, an almost optimal solution can be obtained by discretizing (naively) the state space using the same discretization steps as for action effects. Alternatively, we can maintain a more compact representation of the value function as a piecewise constant function, and try to keep the corresponding state partition as coarse as possible. As described above, we choose the second approach.

As dealing with state partitions in multiple dimensions is difficult and costly, we would like to restrict our attention to partitions using only (hyper-)rectangular pieces, which are relatively easy to store and manipulate. Therefore, we must assume that all the partitions in the problem definition use only rectangular pieces (or can be reduced to such), that is:

- the partition into pieces over which the rewards $R(s, a, s')$ are constant,
- the partition into pieces over which the discrete marginals $T_x(x' | s, a)$ are constant,
- the partition into pieces over which the discretized conditionals $T_{\theta_i}(\theta'_i | s, a, x', \theta'_{<i})$ (absolute) or $T_{\delta_i}(\delta_i | s, a, x', \theta'_{<i})$ (relative) are constant,
- in the case where the actions available vary from one continuous state to another, the partition in pieces over which the set of actions available is constant,

However, this is not sufficient since applying the max operator to two piecewise constant functions may create non-rectangular pieces, even if both arguments contain only rectangular pieces. This may happen in Eqn. 1, where we compare Q -values represented as piecewise constant functions.⁴ In other words, the coarsest partition in a piecewise constant model is not necessarily composed of rectangular pieces, even if all the hypotheses above are true. Therefore,

⁴Note that Eqn. 1 represents a crucial step in the DP algorithm. The max operator implements Bellman’s optimality principle, which is the fundamental principle used to prune the search in the plan space.

we have to abandon either the minimality of the partitions or the restriction to rectangular pieces. In this work, we pursue the first approach and use a max operator that outputs a non-minimal partition made only of rectangular pieces.

We now explain how the rectangular partitions are backed up by our functional DP algorithm. Consider backing up the piecewise constant value function V_{n+1} to compute the function $U_n(s, a, x')$ as in Eqn. 3. Given a state $s = (x, \theta)$, we want to determine the shape and position of the piece of the new value function U_n to which it belongs. For this purpose, we first determine the biggest piece P_s that contains s and over which all parameters of the problem (rewards, discrete marginals, and discretized conditionals) are constant. This is done by intersecting the partitions enumerated above. Then, if action a has an absolute effect on all continuous variables, we stop here: the piece of U_n to which s belongs has the same shape and position as P_s . If a has a relative effect on some variables, then the partition of V_{n+1} may also play a role. As shown in Fig. 1, we must consider all possible arrival states after applying a in s (knowing we end up in x'). Then, for each possible arrival state s' , we calculate how much each coordinate of s' could vary in each direction before s' moves to a different piece in V_{n+1} . We then remember the minima of these values, over all s' . They represent the bounds inside which the starting state s can vary without any of the possible arrival states changing piece. The piece of U_n containing s is then obtained by intersecting P_s with the hyper-rectangle defined by these bounds and centered on s .⁵ Once the shape and position of this piece is determined, its value is computed using Eqn. 3. Then, the computation of the Q -values $Q_n(s, a)$ from the utility function $U_n(s, a, x')$ following Eqn. 2 is straightforward. It requires intersecting the partitions of $U_n(s, a, x')$ for all $x' \in X$. Finally, as stated above, the max operator in Eqn. 1 outputs a new value function V_n that contains only rectangular pieces.

Data Structures

Even if they are made only of rectangular pieces, state-space partitions are cumbersome to store and manipulate. To more efficiently implement the operations of intersecting partitions and merging pieces, rectangular state-space partitions are represented as k -dimensional (kd-) trees (Friedman, Bentley, & Finkel 1977; Naylor, Amanatides, & Thibault 1990). A kd-tree is a multidimensional generalization of the binary tree in which space is recursively split by hyper-planes orthogonal to one axis. It is constructed as follow: the root node r represents the whole continuous state space θ . It is split by an hyperplane h perpendicular to one axis, which induces a binary partition of r in two new regions $r.ge$ and $r.lt$ corresponding to the positive half-

⁵If the reward $R(s, a, s')$ varies as a function of the arrival state s' , then a similar process must be performed to determine how much the starting state can vary without any of the possible arrival states changing piece in the reward function. Equivalently, one may consider the intersection of the partitions of $V_{n+1}(*)$ and $R(s, a, *)$ (instead of V_{n+1} alone) when performing the reasoning of Fig. 1.

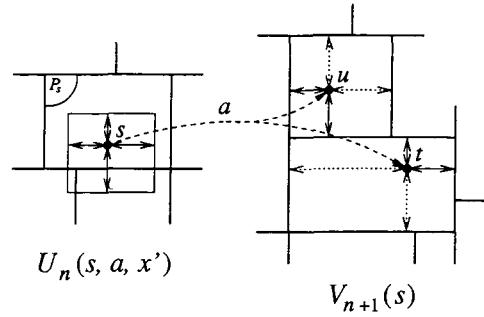


Figure 1: This figure shows how the partition of U_n is computed from the partition of V_{n+1} in the case where there are two continuous variable and the action a has a relative effect on both. The partition on the left side is the partition defined by the data of the problem: The rewards, discrete marginals and discretized conditionals are constant over the piece P_s to which the state s belongs. First we “project” state s through action a , that is we determine all possible arrival states after applying a in s (knowing the arrival discrete state is x'). In this case, there are two possible arrival states denoted by t and u . Then we determine to what piece of V_{n+1} the states t and u belong, and we measure how far t and u are from the borders of their piece in each direction (double arrows). The minima of these distances in each direction (solid double arrows) represent the maximum distance we can move the starting state s without any of the possible arrival states changing pieces. The piece of U_n containing s is obtained by intersecting P_s with the hyper-rectangle defined by these bounds and centered on s .

space and the negative (open) half-space respectively. Each is represented by adding a child node to r . We then repeat this process recursively on the children nodes $r.ge$ and $r.lt$.

The kd-trees facilitate the operations of partition intersection. To compute the intersection of two kd-trees \mathcal{T}_1 and \mathcal{T}_2 defined over the same space, we divide one of them, say \mathcal{T}_1 , at its root. The two sub-trees obtained, $\mathcal{T}_1.ge$ and $\mathcal{T}_1.lt$, will be recursively processed later. We then take the cutting plane $\mathcal{T}_1.h$ of the root of \mathcal{T}_1 and intersect it with the other kd-tree \mathcal{T}_2 . In the simplest implementation, one can perform this operation by constructing a new kd-tree with $\mathcal{T}_1.h$ at the root and two copies of \mathcal{T}_2 as its children. This will be a correct kd-tree. However, we can do better by simplifying each copy of \mathcal{T}_2 . Since each is now defined over a half-space delimited by the cutting plane $\mathcal{T}_1.h$, we can prune each copy by removing subtrees outside the half-space. These subtrees can be identified as child nodes of cuts outside the half-space. Thus we have implemented two routines called **PositiveHalf** and **NegativeHalf** that traverse the tree, compare the region that each tree node represents with the half-space defined by a given cutting plane, and prune away any subtree that is outside of the correct half-plane. To finish the computation of $\mathcal{T}_1 \cap \mathcal{T}_2$, we must recursively intersect the output of these functions with the subtrees $\mathcal{T}_1.ge$ and $\mathcal{T}_1.lt$. The final kd-tree has

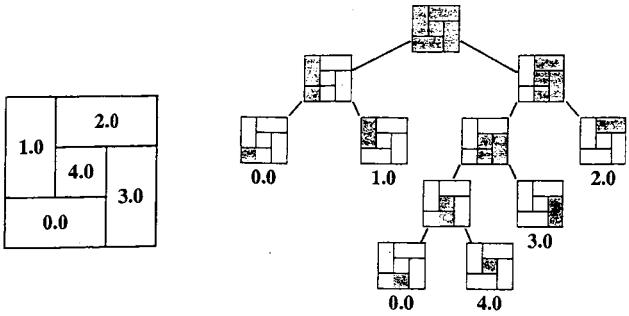


Figure 2: An 2D example of a kd-tree representation of a rectangular partition.

$\mathcal{T}_1.h$ as root, and $\mathcal{T}_1.ge \cap \text{PositiveHalf}(\mathcal{T}_2, \mathcal{T}_1.h)$ and $\mathcal{T}_1.lt \cap \text{NegativeHalf}(\mathcal{T}_2, \mathcal{T}_1.h)$ as children of the root. This process is summarized in the algorithms of Fig. 3. Here, the function `IntersectTerminal` performs the actual computation such as addition or maximization of the data stored at the leaves of the kd-trees.

Partition (\mathcal{T}, h)

1. $\mathcal{R}.h = h$
2. $\mathcal{R}.ge = \text{PositiveHalf}(\mathcal{T}.ge, h)$
3. $\mathcal{R}.lt = \text{NegativeHalf}(\mathcal{T}.lt, h)$
4. **return** \mathcal{R}

`Intersect ($\mathcal{T}_1, \mathcal{T}_2$)`

1. **if** \mathcal{T}_1 or \mathcal{T}_2 is terminal node
2. **return** `IntersectTerminal`($\mathcal{T}_1, \mathcal{T}_2$);
3. $\mathcal{P} = \text{Partition}(\mathcal{T}_2, \mathcal{T}_1.h)$
4. $\mathcal{R}.h = \mathcal{T}_1.h$
5. $\mathcal{R}.lt = \text{Intersect}(\mathcal{T}_1.lt, \mathcal{P}.lt)$
6. $\mathcal{R}.ge = \text{Intersect}(\mathcal{T}_1.ge, \mathcal{P}.ge)$
7. **return** \mathcal{R}

Figure 3: Algorithms for intersecting kd-trees.

Finally, the kd-tree representation is used to support a piece-merging routine for piecewise constant value functions. This routine just performs a depth first exploration of the kd-tree and merges leaf nodes with the same parent and the same value. Although this does not guarantee a minimum rectangular partition, it provides a good trade-off between simplicity and efficiency.

Piecewise Linear Model

The next step is to introduce piecewise linear reward functions into the model. For example, to take into account the illumination of a rock in a planetary rover problem, the value of a picture could vary (piecewise) linearly with the time of the day. Technically, the basic principle is to replace the scalar value attached to each piece of the reward and value functions by affine functions of the continuous variables. An affine function is a linear function plus a constant. It is encoded by a vector of \mathbb{R}^{l+1} containing the l coefficients of the linear function and the constant.

The solutions for the piecewise constant case are not sufficient in the linear model. In multi-dimensional linear

models, value functions are linear hyperplanes with arbitrary orientation. Thus applying the max operator in Eqn. 1 produces the upper envelope of a set of hyperplanes, which results in pieces that may have arbitrary polyhedral shapes, even if all the hypotheses of the previous section hold.

A similar issue arises in the theory of POMDPs. In this model, the value function is defined over the *belief space* as the upper envelope of a set of linear functions (Kaelbling, Littman, & Cassandra 1998; Cassandra, Littman, & Zhang 1997). The best solution techniques for POMDPs do not keep an *explicit* representation of the partition induced by the max over linear functions, but maintain the subset of linear functions (called α -vectors) that are not dominated within the belief space. The α -vectors *implicitly* define a partition of the belief space. Instead of using an explicit max operator, the algorithms implement Bellman's optimality principle by pruning the sets of α -vectors by removing dominated vectors, which represent linear functions that are optimal nowhere in the belief space.

Borrowing from the POMDP theory, we solve the linear model in the following way. We distinguish between pieces of value functions that are created through the max operator of Eqn. 1 and pieces created through another process (intersecting the partitions of immediate rewards, discrete conditionals and continuous marginals, and the process illustrated in Fig. 1). For the latter case, we perform the computation exactly as in the piecewise constant case. That is, we explicitly compute the effects of Eqn. 2 and Eqn. 3 on the partition of the functions Q_n and U_n . As in the piecewise constant case, if the hypotheses above hold and the value function at step V_{n+1} contains only rectangular pieces, then the pieces created at V_n are all rectangular. For the piecewise linear case, instead of having a single real value attached, each of these pieces carries a set of vectors of dimension $l+1$ that represents a set of affine functions. The value function over the piece is defined as the maximum of these affine functions. So, as for POMDPs, the value function is piecewise linear convex over a piece. It implicitly defines a sub-partition of the rectangular piece into polyhedral pieces.

Equations 2 and 3 reduce to manipulation and production of affine functions. For example, when computing Eqn. 3, after having determined the shape and position of the new piece (as in Fig. 1), we must compute its vector set. This piece of U_n projects to the set of pieces in V_{n+1} that the arrival state can possibly belong to. The resulting vector set for U_n is made of a function applied to each element of the cross-product of the vector sets of these pieces. In particular, we choose a vector among those in each possible arrival piece (one vector per piece). These vectors are composed through a simple process that implements Eqn. 3 and outputs a single affine function: (i) each vector is multiplied by the scalar probability of a transition to the piece where it has been taken; (ii) the vectors are summed to reflect the integral in Eqn. 3; (iii) the linear function representing the reward is added to the resulting vector.⁶ A similar process is repeated for all possible choices of vectors at the arrival

⁶This does not represent exactly the order of arithmetics oper-

pieces. All the resulting affine functions are added to the starting piece.

Finally, Eqn. 1 is replaced by a mechanism of vector pruning. To implement Bellman's optimality principle, the value functions are pruned from every affine function that is not optimal somewhere on its piece. In other words, value functions at each step are represented as *minimal* sets of affine functions. Pruning is performed by solving simple linear programs. Given a set of affine functions F defined over some piece, we build the minimal set M by testing each affine functions $f \in F$ against M for complete dominance using the following linear program:

$$\begin{aligned} \text{variables: } & d, \theta_i \in \Theta \\ & \max d \\ \text{s.t. } & f(\Theta) - m(\Theta) > d, \quad \forall m \in M \\ & \theta_i^b \leq \theta_i < \theta_i^t, \quad 1 \leq i \leq l \end{aligned}$$

where θ_i^b and θ_i^t are the boundaries over the i -th continuous variable in the rectangle piece. Let (d^*, Θ^*) be the solution of the linear program. If $d^* \leq 0$, then the affine function f is completely dominated by M and is discarded from F . Otherwise, we remove the vector from F that has maximal value at Θ^* and add it to M . We then repeat this process until F is empty.

Many POMDP algorithms differ only in the way they purge sets of α -vectors. Although they use the same kind of linear program, they differ in the timing of pruning stages. Similarly, different versions of our algorithm could be derived. For instance, we could prune only the state-wise value functions V_n , or the intermediate value Q_n and U_n too. Further research is needed to identify the best algorithm. In our implementation, we prune every intermediate function *à la* Incremental Pruning (Cassandra, Littman, & Zhang 1997).

Simulation Results

We tested our algorithms on the simple Mars rover domain presented in (Bresina *et al.* 2002). We used three variant of this problem with one, two or three continuous variables representing resources. Tables 1 to 3 compare the performance of our piecewise constant functional DP algorithms with the naive discretization approach.

Discretization	Lazy		Naive	
	time(s)	pieces	time(s)	pieces
25000	29.11	15060	17.90	25000
30000	53.02	18029	27.32	30000
40000	138.32	23931	51.84	40000
50000	254.72	29802	84.81	50000
60000	413.97	35636	124.99	60000
75000	723.79	44312	199.19	75000

Table 1: Results in a 1D piecewise constant model.

ations in Eqn. 3. However, given that the rewards are constants over the piece of state-space of interest, the proposed computation is equivalent and more efficient.

Discretization	Lazy		Naive	
	time(s)	pieces	time(s)	pieces
50	0.05	3027	0.46	2500
100	0.21	6070	2.45	10000
150	1.75	33716	9.79	22500
200	2.70	29786	26.28	40000
250	4.21	46709	62.84	62500
300	10.59	56483	129.80	90000
350	12.45	69308	236.74	122500
400	16.62	49211	399.15	160000
450	63.59	142492	629.07	202500
500	81.48	92999	951.89	250000
550	153.58	180679	1396.22	302500

Table 2: Results in a 2D piecewise constant model.

Discretization	Lazy		Naive	
	time(s)	pieces	time(s)	pieces
20	0.05	262	1.74	8000
40	0.53	1961	70.97	64000
60	1.81	3363	790.98	216000
80	4.65	4835	4418.52	512000
100	11.90	2877	> 2hr	—
120	24.15	7408	> 2hr	—
140	67.23	9274	> 2hr	—

Table 3: Results in a 3D piecewise constant model.

As the tables show, our algorithm is slower than the naive approach for the 1D problem, but considerably faster for the 2D and 3D problems, especially as the discretization becomes finer. In the case of the 1D problem, the slower performance is due to the overhead of tree-building rather than the flat representation used in the naive algorithm. This is particularly a problem as our kd-tree algorithm is optimized for balancing multi-dimensional trees, and performs poorly at balancing 1D trees.

For the 2D and 3D problems, the key advantage of our algorithm is the vastly smaller number of pieces that the lazy discretization produces. The savings here easily overcome the extra cost of maintaining the tree structure. In the case of the 50-discretization in Table 2, although the final number of pieces is larger than the naive approach, the early steps of DP are operating on value functions with many fewer pieces, so the overall algorithm still runs much faster. The optimal value function for the 2D problem with 500-discretization is shown in Fig. 4.

As one might expect, the piecewise-linear algorithm runs much more slowly than the piecewise-constant algorithm, or the naive algorithm. On the 2D problem, it runs in 14.4, 108, and 823 seconds respectively for discretizations of 50, 100, and 150 respectively. However if we have a linear model, the piecewise-linear algorithm is solving it exactly, while the naive approach (and a piecewise constant representation of the same problem) are only approximating the solution.

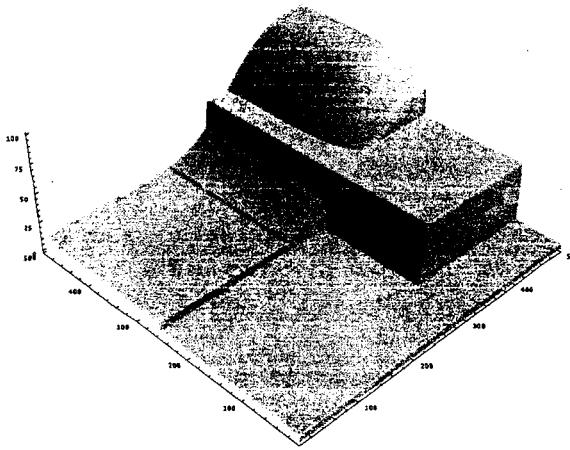


Figure 4: The piecewise-constant value function for the 2D problem.

Conclusions

One important component of Boyan and Littman's model is unaddressed by this work. TMDPs use a particular action called *dawdling* that can be performed for any continuous duration. In a sense, this action accepts a continuous parameter (the duration it should last) that is of the same dimension (seconds) as the continuous variable of the problem (time). There are different ways to generalize the dawdling action in HyMDPs by adding continuous parameters to actions. This is of particular interest for the planetary rover domain. Current planning models used at NASA feature global actions like "drive to landmark 1" that can either succeed if they do not consume more energy than available, or fail if they consume too much. Unfortunately, there is no way in the domain model language to command actions like "drive to landmark 1 but abort in the middle of the drive if the remaining energy falls below threshold t ". However, it is clear that the quality of the plans output would greatly benefit from this ability. Moreover, such a capability would bridge the gap currently existing between the planning models and rover execution languages, such as CRL (Bresina *et al.* 1999), which allow actions to be interrupted when some guard condition fails. Therefore, we will propose models of parameterized actions for HyMDPs in future work.

References

Bertsekas, D., and Tsitsiklis, J. 1996. *Neuro-dynamic Programming*. Belmont, MA: Athena.

Blythe, J. 1999. Decision-theoretic planning. *AI Magazine* 20(2):37–54.

Boutilier, C.; Dearden, R.; and Goldszmidt, M. 2000. Stochastic dynamic programming with factored representations. *Artificial Intelligence* 121:49–107.

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision theoretic planning: structural assumptions and computational leverage. *Journal of AI Research* 11:1–94.

Boyan, J., and Littman, M. 2000. Exact solutions to time-dependent MDPs. In *Advances in Neural Information Processing Systems 13*. Cambridge: MIT Press. 1–7.

Bresina, J.; Golden, K.; Smith, D.; and Washington, R. 1999. Increased flexibility and robustness of mars rovers. In *Proceedings of the Fifth International Symposium on Artificial Intelligence, Robotics and Automation in Space*.

Bresina, J.; Dearden, R.; Meuleau, N.; Ramakrishnan, S.; Smith, D.; and Washington, R. 2002. Planning under continuous time and resource uncertainty: A challenge for AI. In *Proc. of UAI-2002*.

Cassandra, A.; Littman, M.; and Zhang, N. 1997. Incremental Pruning: A simple, fast, exact method for partially observable Markov decision processes. In *Proc. of UAI-97*, 54–61.

Estlin, T.; Fisher, F.; Gaines, D.; Chouinard, C.; Schaffer, S.; and Nesnas, I. 2002. Continuous planning and execution for an autonomous rover. In *Proceedings of the Third International NASA Workshop on Planning and Scheduling for Space*.

Friedman, J.; Bentley, J.; and Finkel, R. 1977. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software* 3(3):209–226.

Howard, R. 1970. *Dynamic Programming and Markov Processes*. Cambridge, MA: MIT Press.

Hyafil, N., and Bacchus, F. 2003. Conformant probabilistic planning via CSPs. In *Proc. of ICAPS-03*, 205–214.

Jónsson, A.; Morris, P.; Muscettola, N.; Rajan, K.; B.; and Smith. 2000. Planning in interplanetary space: theory and practice. In *Proc. of AIPS-2000*, 117–186.

Kaelbling, L.; Littman, M.; and Cassandra, A. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101:99–134.

Meuleau, N., and Smith, D. 2003. Optimal limited contingency planning. In *Proc. of UAI-2003*, 417–426.

Munos, R., and Moore, A. 2002. Variable resolution discretization in optimal control. *Machine Learning* 49:291–323.

Munos, R. 2000. A study of reinforcement learning in the continuous case by the means of viscosity solutions. *Machine Learning* 40:265–299.

Muscettola, N.; Nayak, P.; Pell, B.; and Williams, B. 1998. Remote agent: To boldly go where no AI system has gone before. *Artificial Intelligence* 103(1–2):5–47.

Naylor, B.; Amanatides, J.; and Thibault, W. 1990. Merging BSP trees yields polyhedral set operations. *Computer Graphics (SIGGRAPH'90 Proceedings)* 24(4):115–124.

Puterman, M. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York, NY: Wiley.

Sutton, R., and Barto, A. 1998. *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press.